

# grain

*D Language for Deep Learning*

ML Meetup KANSAI #3 LT

4. Oct. 2018

# D Language for Deep Learning

## language

- ▶ **like C++**: fast, strongly typed, LLVM/GCC backend
- ▶ **like Python**: simple, lightweight, jupyter support

## libraries<sup>1</sup>

- ▶ **mir**: N-dim fast algorithm, numpy-like APIs
- ▶ **dcompute**: CUDA kernel DSL

---

<sup>1</sup><https://github.com/libmir>

grain

## deep learning framework for D

- ▶ <https://github.com/ShigekiKarita/grain>
- ▶ boost software license 1.0

## philosophy

- ▶ **DYNAMIC**: like chainer and pytorch
- ▶ **SAFE**: statically typed variable and function
- ▶ **LIGHT**: simple like Python, small like C++
- ▶ **FAST**: mir and CUDA backend

grain is **dynamic**

like chainer ...

```
1 foreach (epoch; 0 .. 10) {  
2     foreach (i; niter.permutation) {  
3         auto xs = inputs[i].variable;  
4         auto ts = targets[i].variable;  
5         auto ys = model(xs);  
6         auto loss = crossEntropy(ys, ts);  
7         auto acc = accuracy(ys, ts);  
8         model.zeroGrad();  
9         loss.backward();  
10        optimizer.update();  
11    }  
12 }
```

grain is **safe**

but statically typed and optimized.

```
1 foreach (epoch; 0 .. 10) {  
2   foreach (i; niter.permutation) {  
3     Variable!(float, 3, HostStorage) xs = inputs[i].variable;  
4     Variable!(int, 1, HostStorage) ts = targets[i].variable;  
5     Variable!(float, 2, HostStorage) ys = model(xs);  
6     Variable!(float, 0, HostStorage) loss = crossEntropy(ys, ts);  
7     float acc = accuracy(ys, ts);  
8     model.zeroGrad();  
9     loss.backward();  
10    optimizer.update();  
11  }  
12 }
```

grain is **safe**

every function is statically typed and optimized.

```
1 struct Sigmoid(T, size_t dim) {  
2     Variable!(T, dim, HostStorage) y;  
3  
4     nothrow forward(Variable!(T, dim, HostStorage) x) {  
5         auto y = x.sliced.map!(a => tanh(a * 0.5) * 0.5 + 0.5)  
6             .slice.variable(x.requiresGrad);  
7         if (x.requiresGrad) this.y = y;  
8         return y;  
9     }  
10    nothrow backward(Variable!(T, dim, HostStorage) gy) {  
11        auto ys = this.y.sliced;  
12        return slice((1.0 - ys) * ys * gy.sliced).variable;  
13    }  
14    mixin FunctionCommon; // inject type checking  
15 }
```

grain is **safe**

## Chainer/PyTorch issue

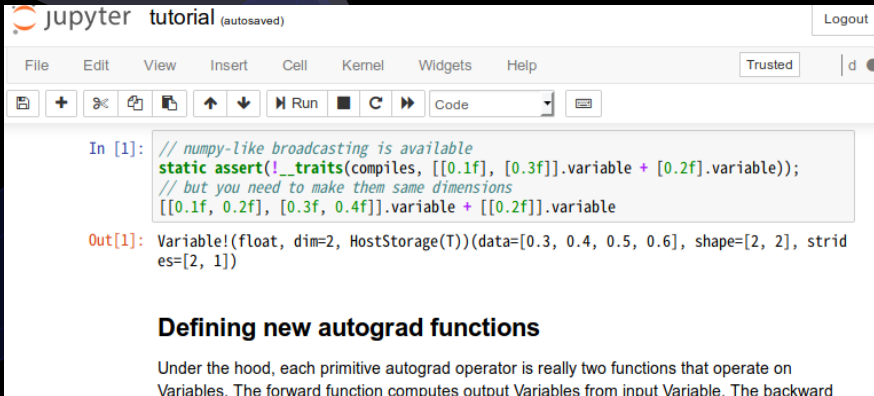
- ▶ runtime overhead
  - ▶ for-loop, dynamic dispatch, func call
- ▶ runtime error:
  - ▶ type error, exception safety, memory leak

## D solution

- ▶ template based code generation
- ▶ compile-time type/error checking

# grain is a lightweight framework

## Jupyter notebook support <sup>2</sup>



The screenshot shows a Jupyter Notebook window titled "tutorial (autosaved)". The interface includes a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help) and a toolbar with icons for saving, adding, deleting, and running cells. The code cell contains the following Python code:

```
In [1]: // numpy-like broadcasting is available  
static_assert(!_traits(compiles, [[0.1f], [0.3f]].variable + [0.2f].variable));  
// but you need to make them same dimensions  
[[0.1f, 0.2f], [0.3f, 0.4f]].variable + [[0.2f]].variable
```

The output of the cell is:

```
Out[1]: Variable!(float, dim=2, HostStorage(T))(data=[0.3, 0.4, 0.5, 0.6], shape=[2, 2], strides=[2, 1])
```

Below the code cell, the text "Defining new autograd functions" is displayed, followed by a paragraph explaining that each primitive autograd operator is implemented as two functions (forward and backward) that operate on Variables.

<sup>2</sup><https://github.com/ShigekiKarita/grain/blob/master/tutorial.ipynb>



grain is a lightweight framework

smaller code and footprint

framework	code lines	lib size [mb]	lib type
<b>grain</b>	<b>12,431</b>	<b>0.6</b>	<b>static</b>
chainer	162,106	6	python code
pytorch	193,754	911	dynamic
tensorflow	130,475	285	dynamic

smaller exe file (MNIST : 1.8MB, CIFAR: 2.3MB)

grain is as **fast** as other frameworks

task	backend	framework	train iter/sec
mnist	CUDA	grain	270
		chainer	340
		pytorch	200
	CPU	grain	160
		chainer	95
		pytorch	110

- ▶ chainer 4.5.0, pytorch 0.4.1, MKL2018, CUDA9, CUDNN7
- ▶ pytorch is built from source. modified official scripts to be fair.

grain is as **fast** as other frameworks

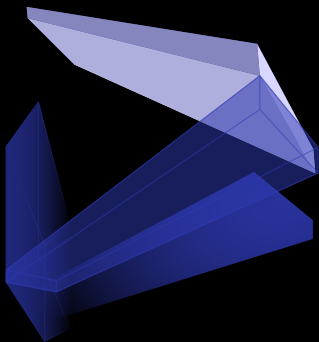
task	backend	framework	train iter/sec
ptb	CUDA	grain	3.1
		chainer	3.4
		pytorch	12
	CPU	grain	1.2
		chainer	2.1
		pytorch	2.4

- ▶ chainer 4.5.0, pytorch 0.4.1, MKL2018, CUDA9, CUDNN7
- ▶ pytorch is built from source. modified official scripts to be fair.

grain: summary

## deep learning framework for **D language**

- ▶ **DYNAMIC**: like chainer and pytorch
- ▶ **SAFE**: statically typed variable and function
- ▶ **LIGHT**: simple like Python, small like C++
- ▶ **FAST**: mir and CUDA backend



Thanks for your attention

<https://github.com/ShigekiKarita/grain>

# examples

- ▶ Image recognition (mnist, cifar)
- ▶ Language modeling (shakespeare, ptb)
- ▶ WIP
  - ▶ Reinforcement learning (cartpole)
  - ▶ Speech recognition (librispeech)
  - ▶ Machine translation (anki)

## future work

- ▶ probabilistic programming
- ▶ lazy evaluation mode
- ▶ low resource environment (RaspberryPi)